REMARKS

Upon entry of the present amendment, claims 1 - 3, 8, 10 - 14, 18, and 20 - 22 will have been amended. In view of the amendments and following remarks, applicants respectfully request reconsideration and withdrawal of each of the outstanding rejections, as well as an indication of the allowability of each of the claims now pending, in due course.

It is noted that the Examiner crossed out the Aptiva Handbook (IBM) reference. An explanation of why the reference was not considered is requested.

In response to the objection to the disclosure, the specification has been amended to add section headings. Thus, it is requested that the objection to the specification be withdrawn.

Claims 10, 12, 20, and 22 have been rejected under 35 U.S.C. §112, second paragraph. The claims have been amended to overcome the §112 rejections. Also, claim 2 is amended to take into account the amendment of claim 1. With regard to claims 12 and 22, these claims now clarify that the extraneous data, program code and execution states are discarded "prior to adding the program code and execution states from said first process to said second sub-process" to provide for the proper antecedent basis referred to in claim 11. Thus, it is requested that the 112 rejections be withdrawn.

In the official action, the Examiner rejected claims 1 and 2 on the ground of obviousness in view of US 6,085,086 (La Porta et. al.). Claims 3 to 22 have been rejected as being obvious in view of the combination of La Porta et. al. and Apple Computers (Technical Introduction to the Macintosh Family). Applicants respectfully traverse.

Regarding claim 1, the Examiner concedes that La Porta et. al. does not explicitly teach using a splitting step to form the first process and the second sub-process. The Examiner argues, however, that because La Porta et. al. teaches using a splitting step to form a sub-process to hold data related to the user process (fork off a child stub process to hold messages sent to the user process, col. 8 lines 44-46), it would be obvious to use the splitting step to form the first process and second sub-process.

The amended claims define the invention more clearly by clarifying that, after splitting, the second sub-process comprises a portion of the program code and/or a portion of the execution state of the computing process not required by the first process. In other words, the program code and/or execution states of the computing process are split into parts, with part of either or both components moved to the second sub-process (those not required by the first process) and the other part remaining in the first process (those still required by the first process). This is particularly advantageous because removing part of the program code and/or execution states reduces the footprint of processes when

resources like memory are scarce, especially when other processes are started on the same host machine.

Normally, a host machine will automatically swap pages of relevant process space in and out of physical memory. Such page swaps, however, are a main contributing factor to the thrashing phenomenon and severe performance degradation. By removing part of the program code and/or execution states, a process can be made to fit within available physical memory, thus avoiding page swaps.

The portion of the program code and/or portion of the execution states removed may be permanent. For example, initialization modules of a process that are needed only once when the process is first started may be removed permanently after the process is started. In the alternative, the portion of the program code and/or portion of the execution states can be removed temporarily and using the same example, the initialization modules can be removed temporarily and re-acquired later, in order to effect swapping at a logical level (e.g., thread of execution, or function) rather than physical pages.

La Porta et. al. relates to a network for providing personal communication services (PCS) for mobile users. The network includes a user process that resides in network nodes to act as an agent for mobile terminals in the PCS environment (column 2 lines 58 to 60). To achieve low call establishment times, the user process is migrated as users move, and the document discloses three embodiments for migrating the user process

(column 3, lines 4 to 15), namely, the Condor approach, the Tern approach, and the Naive approach. The Condor approach is believed to be the most relevant to the present invention.

Specifically, column 8 lines 23 to 63 describes the Condor approach and also referring to Figure 7, the user process 20 includes two main portions, a core 40 and executable 42. The core 40 includes a user area 44, stack 46 and data storage 48, i.e., the core contains the data and the execution states of the parent user process 20. The executable 42 (i.e., the program code) includes text 50, initialized data 52, and symbol and debugging information 54.

Figure 7A and lines 42 to 63 (points 2 to 8) describe the migration steps in further detail, which indicate that the user process 20 first forks off a child stub process that buffers the messages sent to the original user process (point 2). The parent user process 20 then proceeds to save important registers in a buffer and dumps a core (comprising the components of user area 44, stack 46 and data storage 48 as explained earlier), thus exiting. This means that the entire executable portion 42 (i.e., the program code) is not migrated over to the new target USS because the parent user process containing the executable portion 42 exits after dumping the core. Also, it is clear that there is no need to transfer the executable portion 42, because column 7 lines 46 indicates that "all nodes have the executable code for the user process resident" and the new USS uses the "text

segment from the executable present in its node" (see point 5 - column 8 lines 53 and 54), together with the stack and data segments from the old USS (i.e., from the core 40) to create a new user process at the target USS. Accordingly, the core portion 40 dumped by the user process does not include any program code since the entire executable portion 42 remains in the parent user process 20 (contrary to the assertion in the official action against claim 2 that a core dump includes program code) which is not migrated, and because the new USS takes the program code from the executable present at its own node. Thus, the entire execution portion 42 of the user process is intact and is not split up. Similarly, for the execution states stored in the core 40, each component in its entirety is transferred to the new USS, and is not split into parts.

The applicants submit that it would not be obvious to arrive at the present invention in view of the teachings of La Porta et. al. Firstly, the document does not suggest using the "forking" process in place of a "core dump" to form a first process (containing the executable portion 42) and a second process (containing the core 40), and in the absence of such a teaching, the applicant submits that it would not be obvious to do so.

Secondly, even if it was obvious to use the splitting step to form the first and second process, which the applicant submits that it would not be, there is no teaching or suggestion of splitting the user process such that the second sub-process comprises a

portion of the program code and/or a portion of the execution states of the computing process not required by the first process. La Porta et. al. teaches that the child stub process is used to "buffer data" but there is no disclosure or suggestion of how the executables 42 and the core 40 of the parent user process are formed in the child stub process. Thus a skilled artisan would not contemplate splitting up part of the components of the executables (i.e., symbol and debugging info, text, etc.) or part of the components of the core (i.e., the stack, data, etc.). Further, it is clear from column 7 lines 46 that "all nodes have the executable code for the user process resident" because this would "reduce the amount of information that must be transferred between the source and destination nodes, and will in turn reduce the delay in migration" (column 7 lines 46 to 49). A skilled artisan reading this document would thus understand that it would be necessary that each node contains the program code in order to fulfill this consideration. Consequently, the skilled artisan would not be motivated to find ways of performing the splitting of the program code so that a portion of the program code is stored in the core portion 40 and the other part is stored in the executable portion 42, because to do so would go against the teaching of La Porta et al.

Further, La Porta et al. does not teach the splitting of the execution states in the core portion 40 because La Porta et al. teaches that the execution states must be transferred to the new USS to create the new user process and to "continue execution

form where it left off" (see column 8 lines 62 and 63). Consequently, there is no motivation for the skilled artisan reading this document to explore ways of detaching part of the execution states from the core. To do so would not give effect to the invention of La Porta et al., thus rendering it useless because the new USS would not have the required execution states to continue the execution (bearing in mind that the executable portion 42 is not migrated).

As for the Tern approach, the data segment and stack segment of a migrating user process is directly copied to a newly created process, leaving behind the text segment. Thus, each component is copied in its entirety and not in part. Moreover, there is no splitting of a parent process into two sub-processes, unlike the present invention. The Naive approach, on the other hand, adjusts the volume of data (column 10 line 38) and thus is concerned with the manipulation of data, and not program code and/or execution states, unlike what is claimed in the present invention. Thus, the applicants submit that amended claim 1 is not obvious in view of La Porta et. al.

Therefore, in view of the above remarks, the applicants submit that amended claim 1 is patentably distinguished from La Porta et al.

The amendments to the claims add no prohibited new matter. For example, support for the amendments can be found, *inter alia*, at page 3 lines 12 to 18 and page 9 lines 5 to 10.

Dependent claims 2 - 22 are allowable, at least because each depends from allowable independent claim 1, as well as for additional reasons related to their own recitations. Accordingly, for at least the above-noted reasons, Applicants respectfully request reconsideration and withdrawal of the outstanding rejections of claims 1 - 22 as well as an indication of the allowability of each of the claims pending in the present application.

SUMMARY AND CONCLUSION

Applicants believe that the present application is in condition for allowance, and respectfully request an indication to that effect. Applicants have amended the claim set to clarify the features of the present invention. Applicants have also discussed the features recited in applicants' claims and have shown how these features are not taught, disclosed nor rendered obvious by the references applied by the Examiner.

Any amendments to the claims, or addition or cancellation of claims, which have been made in this amendment, and which have not been specifically noted to overcome a rejection based upon the prior art, should be considered to have been made for a purpose unrelated to patentability, and no estoppel should be deemed to attach thereto.

Should the Examiner have any questions, the Examiner is invited to contact the undersigned at the below-listed telephone number.

Respectfully submitted,
Hwee Hwa PANG et al.

Reg. No. 40,163

Bruce H. Bernstein
Reg. No. 29,027

September 20, 2004
GREENBLUM & BERNSTEIN, P.L.C.
1950 Roland Clarke Place
Reston, VA 20191
(703) 716-1191